Московский государственный технический

университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

Выполнил:

студент группы ИУ5-31

Головацкий Андрей

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю. Е.

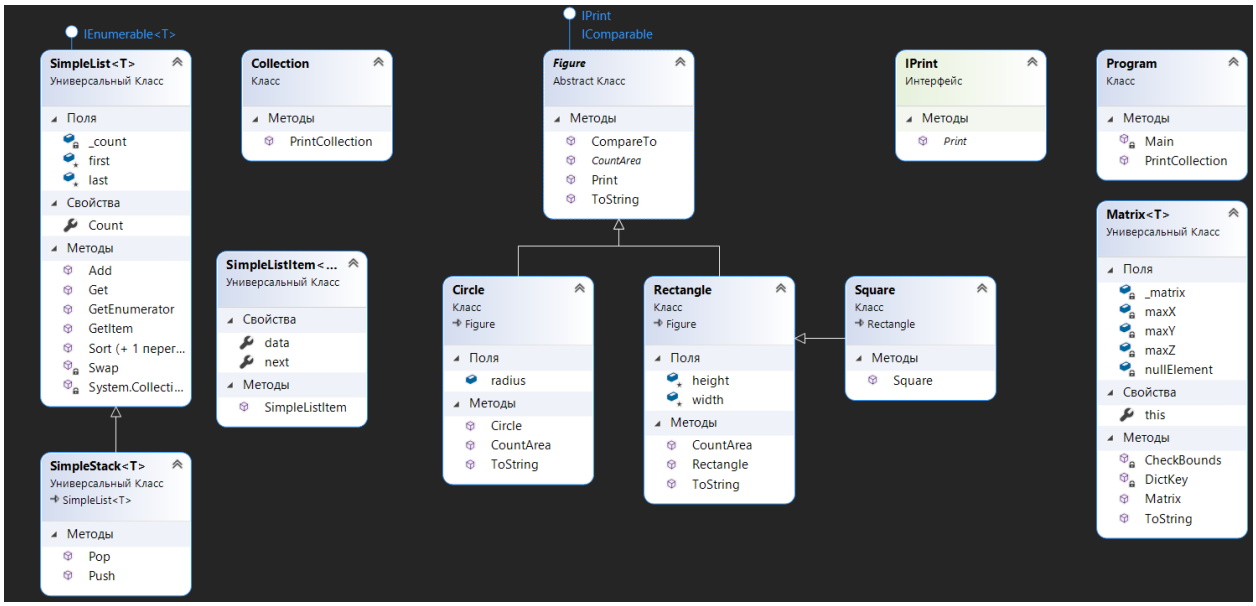
Подпись и дата:

Москва, 2020 г.

Описание задания:

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:



Текст программы:

1) Program.cs:

```
1. using System;
2. using System.Collections;
3. using System.Collections.Generic;
4. using System.Text;
5. using Figures;
6. using Collections;
7. using System.Linq;
8. using SparseMatrix;
9. using FigureCollections;
10.
11.      namespace Collections
12.      {
13.          class Program
14.          {
15.              public static void PrintCollection(IEnumerable col)
16.              {
17.                  foreach (object obj in col)
18.                  {
19.                      if (obj.GetType().Namespace == "Figures")
20.                      {
21.                          Figure t_obj = obj as Figure;
22.                          t_obj.Print();
23.                      }
24.                      else
25.                          Console.WriteLine(obj.ToString());
26.                  }
27.              }
28.              static void Main(string[] args)
29.              {
30.                  Square sq = new Square(5);
31.                  Rectangle rec = new Rectangle(10, 5);
32.                  Circle cir = new Circle(1);
33.                  ArrayList AL1 = new ArrayList();
34.                  AL1.Add(sq);
35.                  AL1.Add(rec);
36.                  AL1.Add(cir);
37.                  AL1.Sort();
```

```
38.                     Console.WriteLine("ArrayList:\n");
39.                     PrintCollection(AL1);
40.                     List<Figure> list1 = new List<Figure>();
41.                     list1.Add(sq);
42.                     list1.Add(rec);
43.                     list1.Add(cir);
44.                     list1.Sort();
45.                     Console.WriteLine("\nList<Figure>:\n");
46.                     PrintCollection(list1);
47.                     Console.Write("\n");
48.                     Matrix<Figure> x = new Matrix<Figure>(4, 3, 3, null);
49.                     x[0, 0, 0] = sq;
50.                     x[1, 1, 0] = rec;
51.                     x[2, 2, 0] = cir;
52.                     Console.WriteLine(x.ToString());
53.                     SimpleStack<Figure> figures = new
    SimpleStack<Figure>();
54.                     figures.Add(cir);
55.                     figures.Add(sq);
56.                     figures.Add(rec);
57.                     Console.WriteLine("Вывод стека:\n");
58.                     for (int i = 0; i < 3; i++)
59.                     {
60.                         Console.Write(figures.Pop().ToString() + "\n");
61.                     }
62.                     Console.WriteLine("\nОбработка ошибки несуществующего
    элемента:\n");
63.                     try
64.                     {
65.                         Console.WriteLine(figures.Pop().ToString());
66.                     }
67.                     catch (Exception e)
68.                     {
69.                         Console.WriteLine(e.Message);
70.                     }
71.                 }
72.             }
73.         }
```

2) Figures.cs:

```
1. using System;
2. 
3. namespace Figures
4. {
5.     public interface IPrint
6.     {
7.         public void Print();
8.     }
9.     public abstract class Figure : IPrint, IComparable
10.         {
11.             public abstract double CountArea();
12.             public int CompareTo(object t_obj)
13.             {
14.                 if (t_obj.GetType().Namespace != "Figures" ||
15.                     this.GetType().Namespace != "Figures")
16.                     throw new ArgumentException("Неверный тип!");
17.                 Figure f_obj = t_obj as Figure;
18.                 if (f_obj.CountArea() > this.CountArea())
19.                 {
20.                     return (-1);
21.                 }
22.                 else if (f_obj.CountArea() == this.CountArea())
23.                     return (0);
24.                 else
25.                     return (1);
26.             }
27.             public override string ToString()
28.             {
29.                 return (this.GetType().Name + " " +
    CountArea().ToString());
30.             }
31.             public void Print()
32.             {
33.                 Console.WriteLine(this.ToString());
34.             }
35.         }
36. 
37.         public class Rectangle : Figure
```

```
38.         {
39.             protected double width;
40.             protected double height;
41.
42.             public Rectangle(double t_width, double t_height)
43.             {
44.                 width = t_width;
45.                 height = t_height;
46.             }
47.             public override double CountArea()
48.             {
49.                 return (width * height);
50.             }
51.             public override string ToString()
52.             {
53.                 return ("{" + this.GetType().Name + ": width: |" +
    width + "| height: |" + height + "| Figure area: |" +
    CountArea().ToString() + "|" + "}");
54.             }
55.         }
56.         public class Square : Rectangle
57.         {
58.             public Square(double t_width) : base(t_width, t_width) { }
59.         }
60.
61.         public class Circle : Figure
62.         {
63.             public double radius;
64.
65.             public Circle(double t_radius)
66.             {
67.                 radius = t_radius;
68.             }
69.
70.             public override double CountArea()
71.             {
72.                 return (Math.PI * radius * radius);
73.             }
74.             public override string ToString()
75.             {
```

```
                return ("{" + this.GetType().Name + ": radius: |" + radius + "| Fiqure area: |" + CountArea().ToString() + "|" + "}");
            }
        }
    }
```

3) Matrix.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SparseMatrix
{
    public class Matrix<T>
    {
        /// <summary>
        /// Словарь для хранения значений
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// <summary>
        /// Количество элементов по горизонтали (максимальное количество столбцов)
        /// </summary>
        int maxX;

        /// <summary>
        /// Количество элементов по вертикали (максимальное количество строк)
        /// </summary>
        int maxY;

        int maxZ;
        /// <summary>
        /// Пустой элемент, который возвращается если элемент с нужными координатами не был задан
        /// </summary>
```

```
29.             T nullElement;
30.
31.             /// <summary>
32.             /// Конструктор
33.             /// </summary>
34.             public Matrix(int px, int py, int pz, T nullElementParam)
35.             {
36.                 this.maxX = px;
37.                 this.maxY = py;
38.                 this.maxZ = pz;
39.                 this.nullElement = nullElementParam;
40.             }
41.
42.             /// <summary>
43.             /// Индексатор для доступа к данных
44.             /// </summary>
45.             public T this[int x, int y, int z]
46.             {
47.                 get
48.                 {
49.                     CheckBounds(x, y, z);
50.                     string key = DictKey(x, y, z);
51.                     if (this._matrix.ContainsKey(key))
52.                     {
53.                         return this._matrix[key];
54.                     }
55.                     else
56.                     {
57.                         return this.nullElement;
58.                     }
59.                 }
60.                 set
61.                 {
62.                     CheckBounds(x, y, z);
63.                     string key = DictKey(x, y, z);
64.                     this._matrix.Add(key, value);
65.                 }
66.             }
67.
68.             /// <summary>
```

```
69.             /// Проверка границ
70.             /// </summary>
71.             void CheckBounds(int x, int y, int z)
72.             {
73.                 if (x < 0 || x >= this.maxX) throw new Exception("x="
    + x + " выходит за границы");
74.                 if (y < 0 || y >= this.maxY) throw new Exception("y="
    + y + " выходит за границы");
75.                 if (z < 0 || z >= this.maxZ) throw new Exception("z="
    + z + " выходит за границы");
76.             }
77.
78.             /// <summary>
79.             /// Формирование ключа
80.             /// </summary>
81.             string DictKey(int x, int y, int z)
82.             {
83.                 return x.ToString() + "_" + y.ToString() + "_" +
    z.ToString();
84.             }
85.
86.             /// <summary>
87.             /// Приведение к строке
88.             /// </summary>
89.             /// <returns></returns>
90.             public override string ToString()
91.             {
92.                 //Класс StringBuilder используется для построения
    длинных строк
93.                 //Это увеличивает производительность по сравнению с
    созданием и склеиванием
94.                 //большого количества обычных строк
95.
96.                 StringBuilder b = new StringBuilder();
97.                 for (int z = 0; z < this.maxZ; z++)
98.                 {
99.                     b.Append("==================> " + (z +
    1).ToString() + " <==================\n\n");
100.                    for (int j = 0; j < this.maxY; j++)
101.                    {
```

```
                        b.Append("[");
                        for (int i = 0; i < this.maxX; i++)
                        {
                            if (i > 0) b.Append("\t");
                            if (this[i, j, z] != null)
b.Append(this[i, j, z].ToString());
                            else b.Append("null");
                        }
                        b.Append("]\n");
                    }
                    if (z != this.maxZ - 1) b.Append("\n\n");
                }
                return b.ToString();
            }

        }
    }
```

4) Stack.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using FigureCollections;

namespace Collections
{
    class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
            public void Push(T element)
            {
                Add(element);
            }
            public T Pop()
            {
                T res;
                if (this.Count == 0)
                {
                    throw new Exception("Попытка считывания
несуществующего элемента!");
```

```
                    }
                    else
                    {
                        if (this.Count == 1)
                        {
                            res = this.first.data;
                            this.first = null;
                            this.last = null;
                        }
                        else
                        {
                            SimpleListItem<T> prev =
   this.GetItem(this.Count - 2);
                            res = this.last.data;
                            this.last = null;
                            this.last = prev;
                        }
                    }
                    this.Count--;
                    return (res);
                }
            }
        }
```

5) SimpleList.cs:

```
using System;
using System.Collections.Generic;

namespace FigureCollections
{
    /// <summary>
    /// Список
    /// </summary>
    public class SimpleList<T> : IEnumerable<T>
                where T : IComparable
            {
                /// <summary>
                /// Первый элемент списка
                /// </summary>
```

```
15.             protected SimpleListItem<T> first = null;
16.
17.             /// <summary>
18.             /// Последний элемент списка
19.             /// </summary>
20.             protected SimpleListItem<T> last = null;
21.
22.             /// <summary>
23.             /// Количество элементов
24.             /// </summary>
25.             public int Count
26.             {
27.                 get { return _count; }
28.                 protected set { _count = value; }
29.             }
30.             int _count;
31.
32.             /// <summary>
33.             /// Добавление элемента
34.             /// </summary>
35.             public void Add(T element)
36.             {
37.                 SimpleListItem<T> newItem = new
    SimpleListItem<T>(element);
38.                 this.Count++;
39.
40.                 //Добавление первого элемента
41.                 if (last == null)
42.                 {
43.                     this.first = newItem;
44.                     this.last = newItem;
45.                 }
46.                 //Добавление следующих элементов
47.                 else
48.                 {
49.                     //Присоединение элемента к цепочке
50.                     this.last.next = newItem;
51.                     //Просоединенный элемент считается последним
52.                     this.last = newItem;
53.                 }
```

```
54.             }
55.
56.             /// <summary>
57.             /// Чтение контейнера с заданным номером
58.             /// </summary>
59.             public SimpleListItem<T> GetItem(int number)
60.             {
61.                 if ((number < 0) || (number >= this.Count))
62.                 {
63.                     //Можно создать собственный класс исключения
64.                     throw new Exception("Выход за границу индекса");
65.                 }
66.
67.                 SimpleListItem<T> current = this.first;
68.                 int i = 0;
69.
70.                 //Пропускаем нужное количество элементов
71.                 while (i < number)
72.                 {
73.                     //Переход к следующему элементу
74.                     current = current.next;
75.                     //Увеличение счетчика
76.                     i++;
77.                 }
78.
79.                 return current;
80.             }
81.
82.             /// <summary>
83.             /// Чтение элемента с заданным номером
84.             /// </summary>
85.             public T Get(int number)
86.             {
87.                 return GetItem(number).data;
88.             }
89.
90.             /// <summary>
91.             /// Для перебора коллекции
92.             /// </summary>
93.             public IEnumerator<T> GetEnumerator()
```

```
94.             {
95.                 SimpleListItem<T> current = this.first;
96.
97.                 //Перебор элементов
98.                 while (current != null)
99.                 {
100.                    //Возврат текущего значения
101.                    yield return current.data;
102.                    //Переход к следующему элементу
103.                    current = current.next;
104.                }
105.            }
106.
107.            //Реализация обобщенного IEnumerator<T> требует реализации
    необобщенного интерфейса
108.            //Данный метод добавляется автоматически при реализации
    интерфейса
109.            System.Collections.IEnumerator
    System.Collections.IEnumerable.GetEnumerator()
110.            {
111.                return GetEnumerator();
112.            }
113.
114.            /// <summary>
115.            /// Сортировка
116.            /// </summary>
117.            public void Sort()
118.            {
119.                Sort(0, this.Count - 1);
120.            }
121.
122.            /// <summary>
123.            /// Алгоритм быстрой сортировки
124.            /// </summary>
125.            private void Sort(int low, int high)
126.            {
127.                int i = low;
128.                int j = high;
129.                T x = Get((low + high) / 2);
130.                do
```

```
131.                 {
132.                     while (Get(i).CompareTo(x) < 0) ++i;
133.                     while (Get(j).CompareTo(x) > 0) --j;
134.                     if (i <= j)
135.                     {
136.                         Swap(i, j);
137.                         i++; j--;
138.                     }
139.                 } while (i <= j);
140.
141.                 if (low < j) Sort(low, j);
142.                 if (i < high) Sort(i, high);
143.             }
144.
145.             /// <summary>
146.             /// Вспомогательный метод для обмена элементов при
    сортировке
147.             /// </summary>
148.             private void Swap(int i, int j)
149.             {
150.                 SimpleListItem<T> ci = GetItem(i);
151.                 SimpleListItem<T> cj = GetItem(j);
152.                 T temp = ci.data;
153.                 ci.data = cj.data;
154.                 cj.data = temp;
155.             }
156.         }
157.     }
```

6) SimpleListItem.cs:

```
1. using System;
2.
3. namespace FigureCollections
4. {
5.     /// <summary>
6.     /// Элемент списка
7.     /// </summary>
8.     public class SimpleListItem<T>
9.     {
```

```
10.                 /// <summary>
11.                 /// Данные
12.                 /// </summary>
13.                 public T data { get; set; }
14.
15.                 /// <summary>
16.                 /// Следующий элемент
17.                 /// </summary>
18.                 public SimpleListItem<T> next { get; set; }
19.
20.                 ///конструктор
21.                 public SimpleListItem(T param)
22.                 {
23.                     this.data = param;
24.                 }
25.             }
26.         }
```

Пример выполнения программы:

```
Консоль отладки Microsoft Visual Studio
ArrayList:

{Circle: radius: |1| Figure area: |3,141592653589793|}
{Square: width: |5| height: |5| Figure area: |25|}
{Rectangle: width: |10| height: |5| Figure area: |50|}

List<Figure>:

{Circle: radius: |1| Figure area: |3,141592653589793|}
{Square: width: |5| height: |5| Figure area: |25|}
{Rectangle: width: |10| height: |5| Figure area: |50|}

==================> 1 <==================

[{Square: width: |5| height: |5| Figure area: |25|}     null    null    null]
[null   {Rectangle: width: |10| height: |5| Figure area: |50|}  null    null]
[null   null    {Circle: radius: |1| Figure area: |3,141592653589793|}  null]


==================> 2 <==================

[null   null    null    null]
[null   null    null    null]
[null   null    null    null]


==================> 3 <==================

[null   null    null    null]
[null   null    null    null]
[null   null    null    null]

Вывод стека:

{Rectangle: width: |10| height: |5| Figure area: |50|}
{Square: width: |5| height: |5| Figure area: |25|}
{Circle: radius: |1| Figure area: |3,141592653589793|}

Обработка ошибки несуществующего элемента:

Попытка считывания несуществующего элемента!
```