Московский государственный технический

университет им. Н.Э. Баумана.

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

«Работа с коллекциями»

Выполнил:

студент группы ИУ5-31Б

Васюнин Михаил Андреевич

Проверил:

преподаватель каф. ИУ5

Гапанюк Юрий Евгеньевич

Подпись и дата:

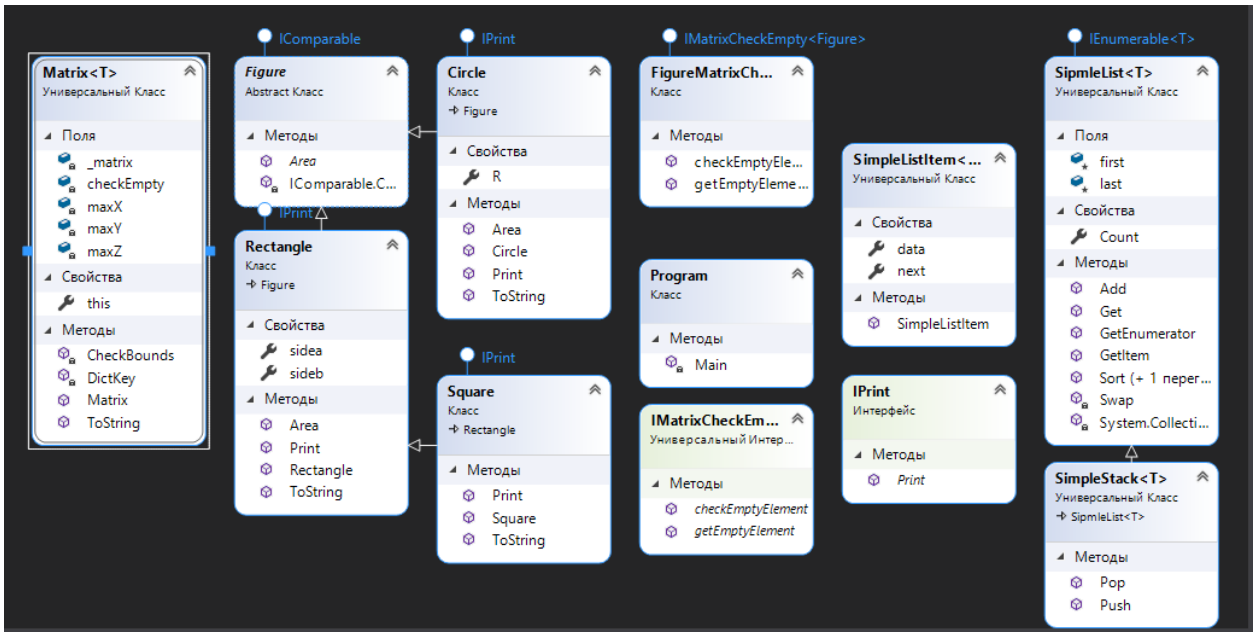
Подпись и дата:

Москва, 2020 г

Постановка задачи

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.

Разработка интерфейса класса



Листинг программы

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

using static System.Math;
namespace Lab3
{
   public interface IMatrixCheckEmpty<T>
   {
      T getEmptyElement();
      bool checkEmptyElement(T element);
   }
   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 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 + " 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++)
      {
        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");
          }
          b.Append("}\n\n");

      }
      return b.ToString();

   }
}

public interface IPrint
{
   void Print();
}

/// <summary>
/// Абстрактный класс фигур
/// </summary>
abstract class Figure : IComparable
{
   public abstract double Area();
   int IComparable.CompareTo(object a2)
   {
      Figure a = (Figure)a2;
      if (this.Area() > a.Area()) return 1;
      else if (this.Area() == a.Area()) return 0;
      else return -1;
   }
}

/// <summary>
/// Прямоугольник
/// </summary>
```

```
    class Rectangle : Figure, IPrint
    {
        public double sidea { get; set; }
        public double sideb { get; set; }
        public Rectangle(int a, int b)
        {
            this.sidea = a;
            this.sideb = b;
        }
        public override double Area()
        {
            return sidea * sideb;
        }
        public override string ToString()
        {
            return "Прямоугольник: длина: " + sidea + ", ширина: " + sideb + ",
площадь: " + sidea * sideb;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }

    /// <summary>
    /// Квадрат
    /// </summary>
    class Square : Rectangle, IPrint
    {
        public Square(int a) : base(a, a) { }
        public override string ToString()
        {
            return "Квадрат: длина стороны: " + sidea + ", площадь: " + sidea * sideb;
        }
        public new void Print()
        {
            Console.WriteLine(this.ToString());
```

```
        }
    }

    /// <summary>
    /// Круг
    /// </summary>
    class Circle : Figure, IPrint
    {
        public int R { get; set; }

        public Circle(int r) { this.R = r; }
        public override double Area()
        {
            return Pow(R, 2) * 3.14;
        }
        public override string ToString()
        {
            return "Круг: радиус: " + R + ", площадь: " + Pow(R, 2) * 3.14;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
    ////////////////////////////////////////////////////////////////////////////
    public class SimpleListItem<T>
    {
        public T data { get; set; }
    public SimpleListItem<T> next { get; set; }
    public SimpleListItem(T param)
    {
        this.data = param;
    }
}
public class SipmleList<T> : IEnumerable<T> where T : IComparable
{
    protected SimpleListItem<T> first = null;
```

```
protected SimpleListItem<T> last = null;
public int Count { get; protected set; }

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;
    }
}



/// ////////////////////////////////////////////////////////////


class SimpleStack<T> : SipmleList<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;
    }
}
//////////////////////////////////////////////////////////
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(5, 4);
            Circle circle = new Circle(16);
            Square square = new Square(18);

            //Необобщённый список
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Необобщённый список");
            Console.ResetColor();
            ArrayList list1 = new ArrayList();
            list1.Add(rect);
            list1.Add(circle);
            list1.Add(square);

            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine("Первая сортировка");
            Console.ResetColor();
            list1.Sort();
            foreach (object o in list1)
                Console.WriteLine(o.ToString());
```

```
        //Обобщенный список
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("\nОбобщённый список");
        Console.ResetColor();
        List<Figure> list2 = new List<Figure>();
        list2.Add(circle);
        list2.Add(rect);
        list2.Add(square);

        Console.ForegroundColor = ConsoleColor.Blue;
        Console.WriteLine("Вторая сортировка");
        Console.ResetColor();
        list2.Sort();
        foreach (object o in list2)
           Console.WriteLine(o.ToString());

        //Разреженная матрица
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("\nРазреженная матрица");
        Console.ResetColor();
        Matrix<Figure> matrix = new Matrix<Figure>(4, 4, 3, new
FigureMatrixCheckEmpty());
        matrix[1, 1, 0] = rect;
        matrix[1, 2, 1] = square;
        matrix[3, 3, 1] = circle;
        Console.WriteLine(matrix.ToString());

        //Стек
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("\nСтек");
        Console.ResetColor();
        SimpleStack<Figure> stack = new SimpleStack<Figure>();
        stack.Add(circle);
        stack.Add(square);
        stack.Add(rect);
```

```
        while (stack.Count > 0)
        {
            Console.WriteLine(stack.Pop());
        }
    }
  }
}
```

Анализ результатов