Защищено:
Гапанюк Ю.Е.

"__"_____2020 г.

Демонстрация:
Гапанюк Ю.Е.

"__"_____2020 г.

Отчет по лабораторной работе № 3
по курсу
Базовые компоненты интернет-технологий

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-52Б _____
(подпись)

Яровенко М. В.

"__"_____2020 г.

Москва, МГТУ - 2020

СОДЕРЖАНИЕ ОТЧЕТА

1. Задание

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.

2. Листинг программы

```
using Microsoft.VisualBasic.CompilerServices;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;
using System.Text;

namespace LR_3._Yarovenko
{
```

```
abstract class Figure : IComparable
{

   public string Name { get; protected set; }

   public abstract double Area();

   public int CompareTo(object obj)
   {
      Figure t = (Figure)obj;
      if (Area() < t.Area()) return -1;
      else if (Area() == t.Area()) return 0;
      else return 1;
   }
}

interface IPrint
{
   void Print();
}

class Rectangle : Figure, IPrint
{
   public double Height { get; protected set; }

   public double Width { get; protected set; }

   public Rectangle(double h, double w)
   {
      Height = h;
      Width = w;
      Name = "Прямоугольник";
   }

   public override double Area()
   {
      double result = Height * Width;
      return result;
   }

   public override string ToString()
   {
```

```
            return "(" + Name + "; Heigth = " + Height + ", Width = " + Width + ", Area
= " + Area().ToString() + ")";
        }

        public void Print() { Console.WriteLine(ToString()); }
    }
    class Square : Rectangle, IPrint
    {
        public Square(double a) : base(a, a) { Name = "Квадрат"; }

        public override string ToString()
        {
            return "(" + Name + "; Heigth = " + Height + ", Width = " + Width + ", Area
= " + Area().ToString() + ")";
        }

    }

    class Circle : Figure, IPrint
    {

        public double Radius { get; protected set; }

        public Circle(double r) { Radius = r; Name = "Круг"; }

        public override double Area()
        {
            double result = Math.PI * Radius * Radius;
            return result;
        }

        public override string ToString()
        {
            return "(" + Name + "; Radius = " + Radius + ", Area = " + Area().ToString()
+ ")";
        }

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

    }
```

```
class Matrix<T>
{

    Dictionary<string, T> _Matrix = new Dictionary<string, T> { };

    int maxZ, maxY, maxX;

    public Matrix(int x, int y, int z) { maxX = x; maxY = y; maxZ = z; }

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

        set
        {
            CheckBounds(x, y, z);
            string key = GetKey(x, y, z);
            _Matrix.Add(key, value);
        }

        get
        {
            CheckBounds(x, y, z);
            string key = GetKey(x, y, z);

            if (_Matrix.ContainsKey(key))
            {
                return _Matrix[key];
            }
            else
            {
                return default(T);
            }

        }

    }

    void CheckBounds(int x, int y, int z)
    {
        if (x<0 || x>=maxX){
            throw new ArgumentOutOfRangeException("x", "x=" + x + " выходит за
границы");
        }
```

```
        if (y<0 || y >= maxY)
        {
            throw new ArgumentOutOfRangeException("y", "y=" + y + " выходит за границы");
        }
        if (z < 0 || z >= maxZ)
        {
            throw new ArgumentOutOfRangeException("z", "z=" + z + " выходит за границы");
        }
    }

    string GetKey(int x, int y, int z) { return x.ToString() + "_" + y.ToString() + "_" + z.ToString(); }


    public override string ToString()
    {
        StringBuilder b = new StringBuilder();
        for (int zz = 0; zz<maxZ; zz++)
        {
            b.Append("|[");
            for (int yy = 0; yy<maxY; yy++)
            {
                if (yy != 0)
                {
                    b.Append("\t[");
                }
                for (int xx = 0; xx < maxX; xx++)
                {

                        if (_Matrix.ContainsKey(GetKey(xx, yy, zz)))
                        {
                            b.Append(this[xx, yy, zz]);
                        }
                        else
                        {
                            b.Append("-");
                        }
                    if (xx < maxX - 1)
                    {
                        b.Append("  ");
                    }
```

```
                }
                b.Append("]");
            }

            b.Append("|\n");
        }
        return b.ToString();
    }
}

public class SimpleListItem<T>
{

    public T data { get; set; }

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

    public SimpleListItem(T param)
    {
        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);
        Count++;
```

```
    if (last == null)
    {
        first = newItem;
        last = newItem;
    }

    else
    {
      last.next = newItem;
      last = newItem;
    }
}

public SimpleListItem<T> GetItem(int number)
{
    if ((number < 0) || (number >= Count))
    {
      throw new Exception("Выход за границу индекса");
    }
    SimpleListItem<T> current = 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 = first;

    while (current != null)
    {

        yield return current.data;
```

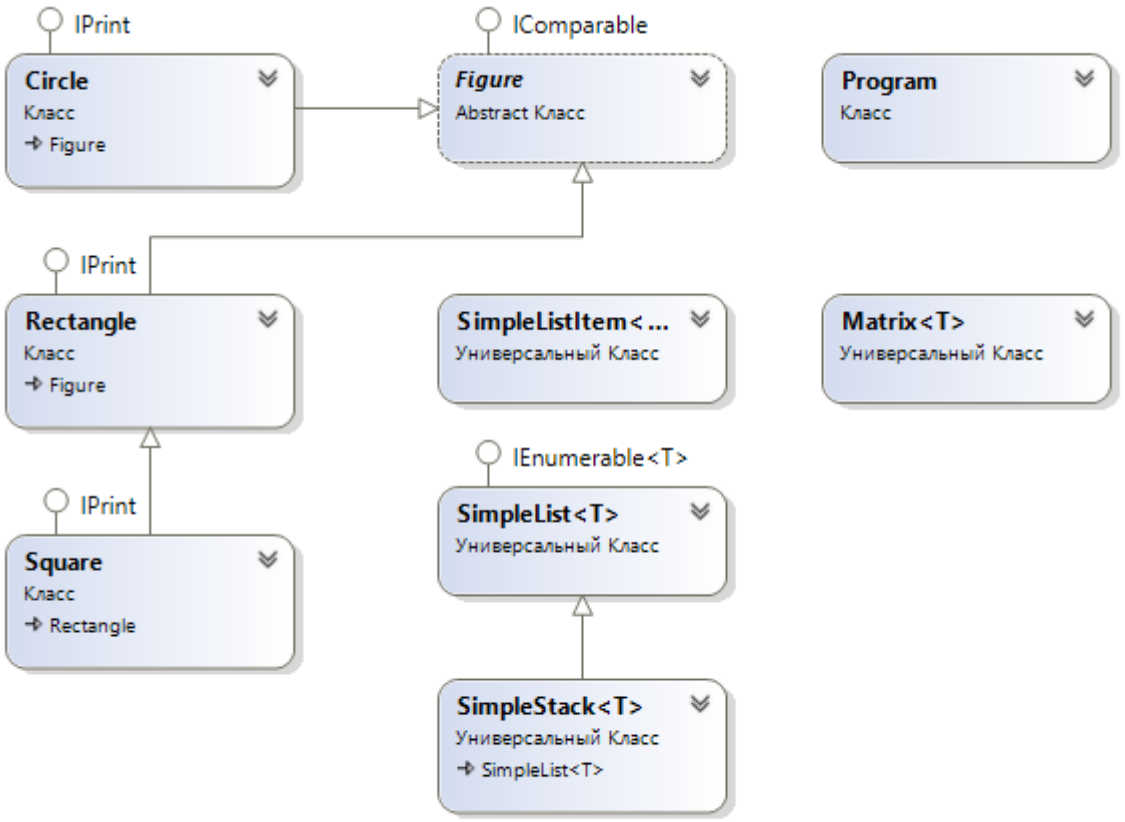
```
        current = current.next;
    }
}


 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> : SimpleList<T> where T : IComparable
    {

        public void Push(T element)
        {
          Add(element);
        }
        public T Pop()
        {

            T Result = default(T);

            if (Count == 0) return Result;

            if (Count == 1)
            {

            Result = first.data;

                this.first = null;
                this.last = null;
            }

            else
            {
                SimpleListItem<T> newLast = GetItem(Count - 2);
                Result = newLast.next.data;
                last = newLast;
               newLast.next = null;
            }
           Count--;

            return Result;
        }
    }
```

```
    class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("ЛР №3. Яровенко Максим, ИУ5Ц-52Б\n");

      Rectangle rec = new Rectangle(2, 5);
      Square squ = new Square(3);
      Circle cir = new Circle(4);

      ArrayList ArL = new ArrayList { rec, squ, cir };
      ArL.Sort();
      Console.WriteLine("Вывод отсортированного необобщенного списка
ArrayList:");
      foreach(object k in ArL)
      {
        Console.WriteLine(k);
      }

      List<Figure> Lst = new List<Figure> { rec, squ, cir };
      Lst.Sort();
      Console.WriteLine("\nВывод отсортированного обобщенного списка
List<Figure>:");
      foreach (object k in Lst)
      {
        Console.WriteLine(k);
      }

      Console.WriteLine("\nРазреженная матрица:\n");

      Matrix<Figure> test = new Matrix<Figure> (3, 3, 5);
      test[0, 0, 0] = rec;
      test[2, 2, 0] = squ;
      test[1, 1, 3] = cir;
      Console.WriteLine(test);

      Console.WriteLine("\nСтек:\n");

      SimpleStack<Figure> stack = new SimpleStack<Figure>();
      stack.Push(rec);
      stack.Push(squ);
```

```
            stack.Push(cir);

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

        }
    }
}
```

3. Диаграмма классов



4. Результаты работы программы