**Отчет по лабораторной работе № 3**

**по курсу «Базовые компоненты интернет технологий»**

Выполнил:

студент группы ИУ5-33

Нагорнов Алексей

Дата: 04.10.2017

**Описание задания лабораторной работы:**

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 Matrix (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») 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 длягеометрических фигур.7. Реализовать класс «SimpleStack» на основе односвязного списка. КлассSimpleStack наследуется от класса SimpleList (представлен в разделе 8

«Вспомогательные материалы для выполнения лабораторных работ»). 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.

**Текст программы на языке C#.**

using System;  
using static System.Math;  
using System.Collections;  
using System.Collections.Generic;  
using System.Text;  
  
namespace SimpleListProject  
{  
    /// <summary>  
    /// Элемент списка  
    /// </summary>  
    public class SimpleListItem<T>  
    {  
        /// <summary>  
        /// Данные  
        /// </summary>  
        public T data { get; set; }  
        /// <summary>  
        /// Следующий элемент  
        /// </summary>  
        public SimpleListItem<T> next { get; set; }  
  
        ///конструктор  
        public SimpleListItem(T param)  
        {  
            this.data = param;  
        }  
    }  
  
    /// <summary>  
    /// Список  
    /// </summary>  
    public class SimpleList<T> : IEnumerable<T>  
        where T : IComparable  
    {  
        /// <summary>  
        /// Первый элемент списка  
        /// </summary>  
        protected SimpleListItem<T> first = null;  
  
        /// <summary>  
        /// Последний элемент списка  
        /// </summary>  
        protected SimpleListItem<T> last = null;  
  
        /// <summary>  
        /// Количество элементов  
        /// </summary>  
        public int Count  
        {  
            get { return \_count; }  
            protected set { \_count = value; }  
        }  
        int \_count;  
  
        /// <summary>  
        /// Добавление элемента  
        /// </summary>  
        /// <param name="element"></param>  
        public void Add(T element)  
        {  
            SimpleListItem<T> newItem = new SimpleListItem<T>(element);  
            this.Count++;  
  
            //Добавление первого элемента  
            if (last == null)  
            {  
                this.first = newItem;  
                this.last = newItem;  
            }  
            //Добавление следующих элементов  
            else  
            {  
                //Присоединение элемента к цепочке  
                this.last.next = newItem;  
                //Просоединенный элемент считается последним  
                this.last = newItem;  
            }  
        }  
  
        /// <summary>  
        /// Чтение контейнера с заданным номером   
        /// </summary>  
        public SimpleListItem<T> GetItem(int number)  
        {  
            if ((number < 0) || (number >= this.Count))  
            {  
                //Можно создать собственный класс исключения  
                throw new Exception("Выход за границу индекса");  
            }  
  
            SimpleListItem<T> current = this.first;  
            int i = 0;  
  
            //Пропускаем нужное количество элементов  
            while (i < number)  
            {  
                //Переход к следующему элементу  
                current = current.next;  
                //Увеличение счетчика  
                i++;  
            }  
  
            return current;  
        }  
  
        /// <summary>  
        /// Чтение элемента с заданным номером  
        /// </summary>  
        public T Get(int number)  
        {  
            return GetItem(number).data;  
        }  
  
  
        /// <summary>  
        /// Для перебора коллекции  
        /// </summary>  
        public IEnumerator<T> GetEnumerator()  
        {  
            SimpleListItem<T> current = this.first;  
  
            //Перебор элементов  
            while (current != null)  
            {  
                //Возврат текущего значения  
                yield return current.data;  
                //Переход к следующему элементу  
                current = current.next;  
            }  
        }  
  
        //Реализация обощенного IEnumerator<T> требует реализации необобщенного интерфейса  
        //Данный метод добавляется автоматически при реализации интерфейса  
        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()  
        {  
            return GetEnumerator();  
        }  
  
        /// <summary>  
        /// Cортировка  
        /// </summary>  
        public void Sort()  
        {  
            Sort(0, this.Count - 1);  
        }  
  
        /// <summary>  
        /// Алгоритм быстрой сортировки  
        /// </summary>  
        /// <param name="low"></param>  
        /// <param name="high"></param>  
        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);  
        }  
  
        /// <summary>  
        /// Вспомогательный метод для обмена элементов при сортировке  
        /// </summary>  
        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;  
        }  
    }  
}  
  
  
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>  
        T nullElement;  
  
        /// <summary>  
        /// Конструктор  
        /// </summary>  
        public Matrix(int px, int py, int pz, T nullElementParam)  
        {  
            this.maxX = px;  
            this.maxY = py;  
            this.maxZ = pz;  
            this.nullElement = nullElementParam;  
        }  
  
        /// <summary>  
        /// Индексатор для доступа к данных  
        /// </summary>  
        public T this[int x, int y, int z]  
        {  
            get  
            {  
                CheckBounds(x, y, z);  
                string key = DictKey(x, y, z);  
                if (this.\_matrix.ContainsKey(key))  
                {  
                    return this.\_matrix[key];  
                }  
                else  
                {  
                    return this.nullElement;  
                }  
            }  
            set  
            {  
                CheckBounds(x, y, z);  
                string key = DictKey(x, y, z);  
                this.\_matrix.Add(key, value);  
            }  
        }  
  
        /// <summary>  
        /// Проверка границ  
        /// </summary>  
        void CheckBounds(int x, int y, int z)  
        {  
            if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за границы");  
            if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за границы");  
            if (z < 0 || z >= this.maxZ) throw new Exception("z=" + z + " выходит за границы");  
        }  
  
        /// <summary>  
        /// Формирование ключа  
        /// </summary>  
        string DictKey(int x, int y, int z)  
        {  
            return x.ToString() + "\_" + y.ToString() + "\_" + maxZ.ToString();  
        }  
  
        /// <summary>  
        /// Приведение к строке  
        /// </summary>  
        /// <returns></returns>  
        public override string ToString()  
        {  
            //Класс StringBuilder используется для построения длинных строк  
            //Это увеличивает производительность по сравнению с созданием и склеиванием   
            //большого количества обычных строк  
  
            StringBuilder b = new StringBuilder();  
  
            for (int k = 0; k < this.maxZ; k++)  
            {  
  
                for (int j = 0; j < this.maxY; j++)  
                {  
                    b.Append("[");  
                    for (int i = 0; i < this.maxX; i++)  
                    {  
                        if (i > 0) b.Append("\t");  
                        b.Append(this[k, i, j].ToString());  
                    }  
                    b.Append("]\n");  
                }  
                b.Append("\n");  
            }  
            return b.ToString();  
        }  
  
    }  
}  
  
  
namespace Лаба\_2  
{  
    class SimpleStack<T> : SimpleListProject.SimpleList<T> where T : IComparable  
    {  
        public void Push(T element)  
        {  
            Add(element);  
        }  
        public T Pop()  
        {  
            SimpleListProject.SimpleListItem<T> itemPopped = last;  
            Count--;  
            if (Count == 0)  
            {  
                last = null;  
                first = null;  
            }  
            else  
            {  
                SimpleListProject.SimpleListItem<T> newLastItem = this.GetItem(Count - 1);  
                last = newLastItem;  
                last.next = null;  
            }  
            return itemPopped.data;  
        }  
    }

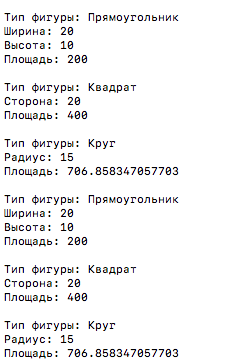
    interface IPrint  
    {  
        void print();  
    }

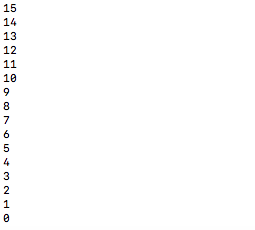
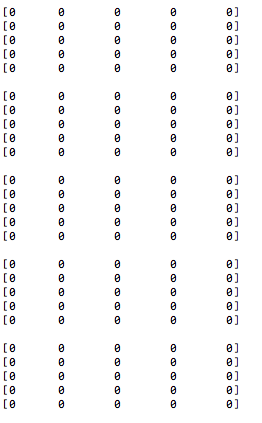
    abstract class GeometricFigure : IPrint, IComparable  
    {  
        public string type { get; set; }  
        public abstract double area();  
        public override string ToString()  
        {  
            return String.Format("Тип фигуры: {0} \n", type);  
        }  
  
        public void print()  
        {  
            Console.WriteLine(this);  
        }  
  
        public int CompareTo(Object otherObj)  
        {  
            GeometricFigure obj = otherObj as GeometricFigure;  
            if(Abs(this.area() - obj.area()) > 0.001)  
            {  
                if(this.area() > obj.area())  
                {  
                    return 1;  
                }  
                else  
                {  
                    return -1;     
                }  
            }  
            else   
            {  
                return 0;  
            }  
        }  
    }

    class Rectangle : GeometricFigure  
    {  
        public double height { get; set; }  
        public double width { get; set; }  
  
        public Rectangle(double \_height, double \_width)  
        {  
            height = \_height;  
            width = \_width;  
            type = "Прямоугольник";  
        }  
  
        public override double area()  
        {  
            return height \* width;  
        }  
        public override string ToString()  
        {  
            return base.ToString() + String.Format("Ширина: {0}\nВысота: {1}\nПлощадь: {2}\n", width, height, area());  
  
        }  
    }  
  
    class Square : Rectangle  
    {  
        public double side { get; set; }  
  
        public Square(double side) : base(side, side)  
        {  
            type = "Квадрат";  
        }  
  
        public override double area()  
        {  
            return Pow(width, 2);  
        }  
        public override string ToString()  
        {  
            return String.Format("Тип фигуры: {0}\nСторона: {1}\nПлощадь: {2}\n", type, width, area());  
        }  
    }

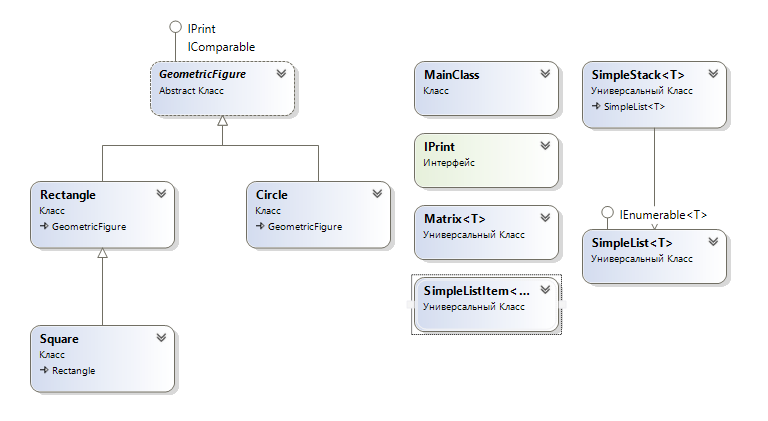
    class Circle : GeometricFigure  
    {  
        public double radius { get; set; }  
  
        public Circle(double \_radius)  
        {  
            radius = \_radius;  
            type = "Круг";  
        }  
  
        public override double area()  
        {  
            return PI \* Pow(radius, 2);  
        }  
        public override string ToString()  
        {  
            return base.ToString() + string.Format("Радиус: {0}\nПлощадь: {1}\n", radius, area());  
        }  
    }

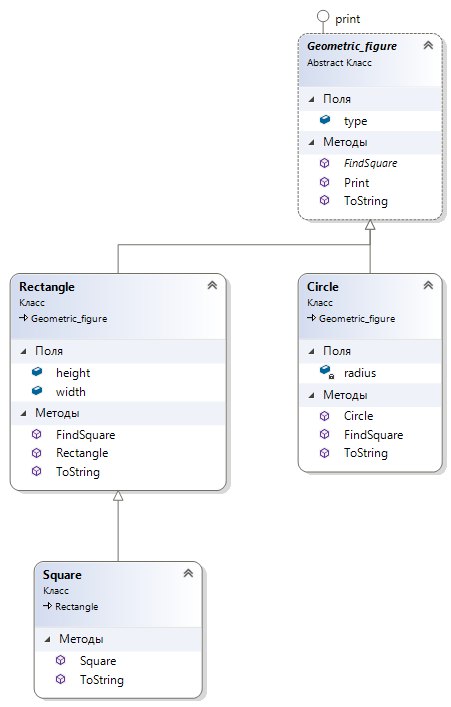
    class MainClass  
    {  
        public static void Main(string[] args)  
        {  
            Rectangle rec = new Rectangle(10, 20);  
            Square sq = new Square(20);  
            Circle circ = new Circle(15);  
            ArrayList list = new ArrayList();  
  
            list.Add(rec);  
            list.Add(sq);  
            list.Add(circ);  
            list.Sort();  
            foreach (GeometricFigure figure in list)  
            {  
                figure.print();  
            }  
  
            List<GeometricFigure> genericList = new List<GeometricFigure>();  
            genericList.Add(rec);  
            genericList.Add(sq);  
            genericList.Add(circ);  
            genericList.Sort();  
            foreach (GeometricFigure figure in genericList)  
            {  
                figure.print();  
            }  
  
            SparseMatrix.Matrix<int> matr = new SparseMatrix.Matrix<int>(5, 5, 5, 0);  
  
            Console.WriteLine(matr);  
  
            SimpleStack<int> stack = new SimpleStack<int>();  
            for (int i = 0; i <= 15; i++)  
            {  
                stack.Push(i);  
            }  
            for (int i = stack.Count; i > 0; i--)  
            {  
                Console.WriteLine(stack.Pop());  
            }  
        }  
  
    }  
}

**Результаты выполнения программы:**

****

**Диаграмма классов:**

****

****